Intel® Celeron™ Processor Specification Update

Release Date: May 13, 1998

Order Number: 243748-002

The Intel® Celeron™ processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY v
PREFACE vi
Specification Update for Intel® Celeron™ Processors
GENERAL INFORMATION 3
ERRATA 9
DOCUMENTATION CHANGES 28
SPECIFICATION CLARIFICATIONS 31

REVISION HISTORY

Date of Revision	Version	Description
April 15, 1998	-001	This document is the first Specification Update for the Intel® Celeron™ processor.
May 13, 1998	-002	Added Errata 24 through 28.

PREFACE

This document is an update to the specifications contained the *Intel® Celeron™ Processor at 266 MHz* datasheet (Order Number 243658), and the *Intel Architecture Software Developer's Manual, Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

Specification Changes are modifications to the current published specifications for the Intel® Celeron™ processor. These changes will be incorporated in the next release of the specifications.

S-Specs are exceptions to the published specifications, and apply only to the units assembled under that s-spec.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Errata are design defects or errors. Errata may cause the Intel Celeron processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor stepping must assume that all errata documented for that processor stepping are present on all devices.

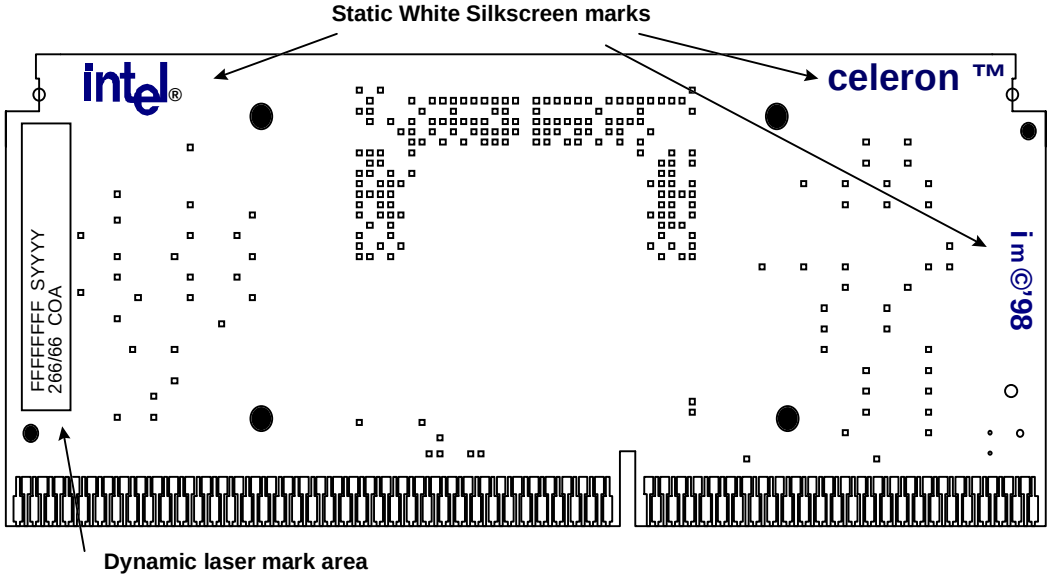
Identification Information

Complete identification information of the Intel Celeron processor can be found in the *Intel Processor Identification and the CPU Instruction* application note (Order Number 241618).

Specification Update for Intel® Celeron™ Processors

GENERAL INFORMATION

Intel® Celeron™ Processor Markings



NOTES:

- SYYYY = S-spec Number.
- FFFFFFFF = FPO # (Test Lot Traceability #).
- COA = Country of Assembly.

Intel® Celeron™ Processor Identification Information

CPUID								
Type	**Family**	**Model**	**Stepping**	**Core Stepping**	**S-Spec**	**Speed (MHz) Core/Bus**	**S.E.C. Substrate**	**Notes**
0	6	5	0	dA0	SL2SY	266/66	Rev. 1	
0	6	5	1	dA1	SL2TR	266/66	Rev. 1	
0	6	5	1	dA1	SL2QG	266/66	Rev. 1	1

NOTE:

1. This is a boxed Intel® Celeron™ processor with an attached fan heatsink.

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the Intel Celeron processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
SUB:	This column refers to errata on the Intel® Celeron™ processor substrate.
Shaded:	This erratum is either new or modified from the previous version of the document.

NO.	dA0	dA1	SUB	Plans	ERRATA
1	X	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
2	X	X		NoFix	Differences exist in debug exception reporting
3	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
4	X	X		NoFix	FP inexact-result exception flag may not be set
5	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
6	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
7	X	X		NoFix	Branch traps do not function if BTMs are also enabled
8	X	X		NoFix	Machine Check Exception handler may not always execute successfully
9	X	X		NoFix	LBER may be corrupted after some events
10	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
11	X	X		Fix	Potential early deassertion of LOCK# during split-lock cycles
12	X	X		NoFix	A20M# may be inverted after returning from SMM and Reset
13	X	X		Fix	Reporting of floating-point exception may be delayed
14	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
15	X	X		Fix	Built-in self test always gives nonzero result
16	X	X		Fix	THERMTRIP# may not be asserted as specified
17	X			Fix	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
18	X			Fix	Snoop cycle generates spurious machine check exception
19	X	X		Fix	MOVD/MOVQ instruction writes to memory prematurely

NO.	dA0	dA1	SUB	Plans	ERRATA
20	X	X		NoFix	Memory type undefined for nonmemory operations
21	X	X		NoFix	Bus protocol conflict with optimized chipsets
22	X	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
23	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
24	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
25	X	X		NoFix	Read portion of RMW instruction may execute twice
26	X	X		Fix	Test pin must be high during power up
27	X	X		Fix	Intervening writeback may occur during split-lock
28	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
1AP	X	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
2AP	X	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

NO.	dA0	dA1	SUB	Plans	DOCUMENTATION CHANGES
1	X	X		Doc	Invalid arithmetic operations and masked responses to them relative to FIST/FISTP instruction
2	X	X		Doc	FIDIV/FIDIVR m16int description
3	X	X		Doc	PUSH does not pad with zeros
4	X	X		Doc	DR7, bit 10 is reserved
5	X	X		Doc	Cache and TLB description correction
6	X	X		Doc	SMRAM state save map contains documentation errors

NO.	dA0	dA1	SUB	Plans	SPECIFICATION CLARIFICATIONS
1	X	X		Doc	Writes to WC memory

NO.	dA0	dA1	SUB	Plans	SPECIFICATION CLARIFICATIONS
2	X	X		Doc	Multiple processors protocol and restrictions
3	X	X		Doc	Critical sequence of events during a page fault exception
4	X	X		Doc	Performance-monitoring counter issues
5	X	X		Doc	POP[ESP] with 16-bit stack size

ERRATA

1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved in 32-bit mode, the subtraction routine used to calculate the FP Data Operand Pointer will assume the floating-point access was in 32-bit mode, and the high word of the address will be FFFFh instead of 0000h.
IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load which wraps the 64-Kbyte boundary is executed.
- The operating system uses a 32-bit handler on an unmasked exception which occurs during the load.
- The exception handler uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.
WORKAROUND: If the FP Data Operand Pointer is used in a 32-bit exception handler in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Intel Celeron processor, as described below:

CASE 1:
The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The Pentium® processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the Pentium processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Intel Celeron processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:
In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:
If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:
If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

IMPLICATION: When debugging or when developing debuggers for a Intel Celeron processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4 (no workaround has been identified for this case).

WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are **disabled** (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register (s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers before they become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

4. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int

- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting a NOP instruction between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

5. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

6. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Intel Celeron processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the

SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and SHUTDOWN of the processor.

WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

7. Branch Traps Do Not Function If BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

IMPLICATION: The branch traps and branch trace message debugging features cannot be used together.

WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

8. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

IMPLICATION: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the

processor to enter SHUTDOWN. Therefore, leaving MCEs disabled may not improve overall system behavior.
WORKAROUND: No workaround which would guarantee successful MCE handler execution under this condition has been identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

9. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the reinitialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.
IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

10. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.
IMPLICATION: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited

during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

11. Potential Early Deassertion of LOCK# During Split-Lock Cycles

PROBLEM: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.
IMPLICATION: This may affect chipset logic which monitors the behavior of LOCK# deassertion.
WORKAROUND: None identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

12. A20M# May Be Inverted After Returning From SMM and Reset

PROBLEM: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.

3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

IMPLICATION: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

WORKAROUND: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

13. Reporting of Floating-Point Exception May Be Delayed

PROBLEM: The Intel Celeron processor normally reports a floating-point exception for an instruction when the next floating-point or MMX™ technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction **after** the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX™ technology instruction, any one of the following occurs:
   a. A subsequent data access occurs to a page which has not been marked as accessed, or
   b. Data is referenced which crosses a page boundary, or
   c. A possible page-fault condition is detected which, when resolved, completes without faulting.
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

IMPLICATION: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible

on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

WORKAROUND: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

14. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP **after** the return value is pushed on the stack, not the value in the ESP register **before** the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an **indirect** call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

15. Built-in Self Test Always Gives Nonzero Result

PROBLEM: The Built-in Self Test (BIST) of the Intel Celeron processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.
IMPLICATION: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.
WORKAROUND: Mask bit 6 of the BIST result register when analyzing BIST results.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

16. THERMTRIP# May Not Be Asserted As Specified

PROBLEM: THERMTRIP# is a signal on the Intel Celeron processor which is asserted when the core reaches a critical temperature during operation as detailed in the processor specification. The Intel Celeron processor may not assert THERMTRIP# until a much higher temperature than the one specified is reached.
IMPLICATION: The THERMTRIP# feature is not functional on the Intel Celeron processor. Note that this erratum can only occur when the processor is running with a T_{PLATE} temperature over the maximum specification of 75 °C.
WORKAROUND: Avoid operation of the Intel Celeron processor outside of the thermal specifications defined by the processor specifications.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

17. Cache State Corruption in the Presence of Page A/D-Bit Setting and Snoop Traffic

PROBLEM: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.
IMPLICATION: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

18. Snoop Cycle Generates Spurious Machine Check Exception

PROBLEM: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines. The processor generates a spurious machine check exception as a result of this unnecessary parity check.
IMPLICATION: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

19. MOVD/MOVQ Instruction Writes to Memory Prematurely

PROBLEM: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.
4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.
IMPLICATION: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX™ instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.
2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
4. Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

WORKAROUND: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Developer's Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

20. Memory Type Undefined for Nonmemory Operations

PROBLEM: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.
IMPLICATION: Bus agents may decode a non-UC memory type for nonmemory bus transactions.
WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.
STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

21. Bus Protocol Conflict With Optimized Chipsets

PROBLEM: A “dead” turnaround cycle with no agent driving the address, address parity, request command, or request parity signals must occur between the processor driving these signals and the chipset driving them after asserting BPRI#. The Intel Celeron processor does not follow this protocol. Thus, if a system uses a chipset or third party agent which optimizes its arbitration latency (reducing it to 2 clocks when it observes an active (low) ADS# signal and an inactive (high) LOCK# signal on the same clock that BPRI# is asserted (driven low)), the Intel Celeron processor may cause bus contention during an unlocked bus exchange.
IMPLICATION: This violation of the reduced arbitration latency bus exchange protocol may cause a system-level setup timing violation on the address, address parity, request command, or request parity signals on the system bus. This may result in a system hang or assertion of the AERR# signal, causing spurious corrective action or shutdown of the system, as the system hardware and software dictate. The possibility of failure due to the contention caused by this erratum may be increased due to the processor’s internal active pull-up of these signals on the clock after the signals are no longer being driven by the processor.
WORKAROUND: If the chipset and third party agents used with the Intel Celeron processor do not optimize their arbitration latency as described above, no action is required. For the 66 MHz Intel Celeron processor, no action is required.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

22. FP Data Operand Pointer May Not Be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.
IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.
WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

23. MOVD Following Zeroing Instruction Can Cause Incorrect Result

PROBLEM: An incorrect result may be calculated after these circumstances:
1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits,
3. This register is then copied to an MMX™ register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX register. Only the MMX register is affected by this erratum.
The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.
1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address>
   or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address>
   or CBW
3. MOVD MM0, EAX

Note that this erratum may occur with “EAX” replaced with any 32-bit general purpose register, and “AX” with the corresponding 16-bit version of that replacement. “BL” or “BX” can be replaced with any 8-bit or 16-bit general purpose register. The CBW instruction is specific to the EAX register only.
In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX or CBW instruction is negative, i.e., bit 15 of AX is a 1.
When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX or CBW instruction.
IMPLICATION: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.
WORKAROUND: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX™ for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly unobtrusive. Again, EAX can be any 32-bit register.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

24. Premature Execution of a Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

With any of the above circumstances, it is possible that the load portion of the instruction will have prematurely executed before the exception handler is entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side effect, restarting the instruction may cause unexpected system behavior.

WORKAROUND: Code that loads from memory that has side effects can effectively workaround this behavior by using simple integer based load instructions when accessing side effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading to side effect memory.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

25. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Intel Celeron processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

IMPLICATION: This erratum has no effect if the memory targeted for the RMW instruction causes no side effects, other than the effect that the memory location is actually read twice. If, however, the load targets a memory region that is associated with read side-effects, multiple occurrences of the initial load may cause unpredictable system behavior.

WORKAROUND: Hardware and software developers who write device drivers for custom hardware which may use a side-effect style of design should use simple loads and simple stores to transfer data to and from the device.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

26. Test Pin Must Be High During Power Up

PROBLEM: The Intel Celeron processor uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, if the TESTHI signal is at a low voltage level when the core power supply comes up, it will cause the processor to enter an invalid test state.
IMPLICATION: If this erratum occurs, the system may boot normally however, L2 cache may not be initialized.
WORKAROUND: Ensure that the 2.5 V($V_{CC2.5}$) power supply ramps at or before the 2.0 V(V_{CCCORE}) power plane. If 2.5 V ramps after core, pull up TESTHI to 2.5 V($V_{CC2.5}$) with a 100K ohm resistor. The internal pull-up will keep the signal from being asserted during power up. Alternatively, TESTHI could be pulled up to 2.0 V(V_{CCCORE}) using a 1K-10K ohm resistor.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

27. Intervening Writeback May Occur During Split-Lock

PROBLEM: During a transaction which has the LOCK# signal asserted and crosses a cache-line boundary (a.k.a. a split-lock transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a split-lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of
the split-lock protocol.
IMPLICATION: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that split-lock transactions may only consist of a read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the

intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.
WORKAROUND: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a split-lock transaction.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

28. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guid,e* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.
IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.
WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

PROBLEM: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter SHUTDOWN after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Intel Celeron processor to enter SHUTDOWN.
IMPLICATION: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.
WORKAROUND: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

PROBLEM: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Intel Celeron processor despite the attempt to mask it via the LVT.

IMPLICATION: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

WORKAROUND: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Intel® Celeron™ Processor at 266 MHz* datasheet and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Documentation Changes will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

Note: The *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3,* applies to all P6 family processors, and therefore some of the documentation changes in this section may not pertain to the Intel Celeron processor specifically.

1. Invalid Arithmetic Operations and Masked Responses to Them Relative to FIST/FISTP Instruction

The *Intel Architecture Software Developer's Manual, Volume 1*, Table 7-20 show "Invalid Arithmetic Operations and the Masked Responses to Them." The table entry corresponding to the FIST/FISTP condition is missing, and is shown below:

Condition	Masked Response
FIST/FISTP instruction when input operand <> MAXINT for destination operand size.	Return MAXNEG to destination operand.

When FIST/FISTP instruction is executed with input operand <> and the destination operand size is MAXINT, the floating-point zero-divide exception will return MAXNEG to the destination operand as its masked response.

2. FIDIV/FIDIVR m16int Description

The *Intel Architecture Software Developer's Manual, Volume 1*, pages 3-118 and 3-122, show in the Description column for the FIDIV *m16int* instruction as "Divide ST(0) by *m64int* by ST(0) and store the result in ST(0)" and FIDIVR *m16int* instruction as "Divide *m64int* by ST(0) and store the result in ST(0)" In both of these cases, *m64int* should be replaced with *m16int*.

3. PUSH Does Not Pad With Zeros

The *Intel Architecture Software Developer's Manual, Volume 1*, page 4-3, contains a section regarding stack alignment. The last sentence in the first paragraph of this section, which reads "If a 16-bit value is pushed onto a 32-bit wide stack, the value is

automatically padded with zeros out to 32-bits." should be removed. The PUSH instruction does not pad with zeros.

4. DR7, Bit 10 is Reserved

The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, shows Figure 14-1, "Debug Registers." Bit 10 of DR7 should be "Reserved" instead of "1."

5. Cache and TLB Description Correction

In the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Table 3-7, the correct description for descriptor value 02H should be as follows:

Descriptor Value	Cache or TLB Description
02H	Instruction TLB: 4M-Byte Pages, **fully** associative, **2** entries

Also, the third bullet after the table should be as follows:

I. Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:
   - 01H–A 32-entry instruction TLB (4-way set associative) for mapping 4-Kbytes pages
   - 02H–A **2**-entry instruction TLB (**fully** associative) for mapping 4-Mbyte pages
   - 03H–A 64-entry data TLB (4-way set associative) for mapping 4-Kbyte pages

For the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Table 9-1, the following corrections should be made:

Cache or Buffer	Characteristics
Instruction TLB (Large Pages)	- P6 family processors: **2** entries, **fully** associative - Pentium® processor: Uses same TLB as used for 4-Kbyte pages - Intel486™ processor: None (large pages not supported)

6. SMRAM State Save Map Contains Documentation Errors

In the *Intel Architecture Software Developer's Manual, Volume 3, System Programming Guide*, Chapter 11, "System Management Mode," Table 11-1 incorrectly documents the SMBASE+Offset for IDT Base and GDT Base for Intel Celeron processors.

The storage locations for these parameters are model specific (i.e., they may differ between the Pentium processor, the Pentium Pro processor, Pentium II processor, Intel Celeron processor, and other P6 family proliferations). These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Intel® Celeron™ Processor at 266 MHz* datasheet and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Specification Clarifications will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

Note: The *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3,* applies to all P6 family processors, and therefore some of the specification clarifications in this section may not pertain to the Intel Celeron processor specifically.

1. Writes to WC Memory

Section 9.3 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, identifies that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses." This sentence should state that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses. The writes may be delayed until the next occurrence of a buffer or processor serialization event, e.g., CPUID execution, a read or write to uncached memory, interrupt occurrence, LOCKed instruction execution, etc., if the WC buffer is partially filled."

2. Multiple Processors Protocol and Restrictions

Section 7.6.1. of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, contain inconsistencies which will be clarified as follows:

7.6.1 Protocol Requirements and Restrictions

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided on all systems based on the Pentium® Pro processor.
- All interrupt mechanisms must be disabled for the duration of the MP protocol algorithm including the window of time between the assertion of INIT# or receipt of an INIT IPI by the application processors and the receipt of a STARTUP IPI by the application processors. That is, requests generated by interrupting devices must not be seen by the local APIC unit (on board the processor) until the completion of the algorithm. Failure to disable the interrupt mechanisms may result in processor shutdown.

- The MP protocol should be initiated only after a hardware reset. After completion of the protocol algorithm, a flag is set in the APIC base MSR of the BSP (APIC_BASE.BSP) to indicate that it is the BSP. This flag is cleared for all other processors. If a processor or the system is subject to an INIT sequence (either through the INIT# pin or an INIT IPI), then the MP protocol is not re-executed. Instead, each processor examines its BSP flag to determine whether the processor should boot or wait for a STARTUP IPI.

3. Critical Sequence of Events During a Page Fault Exception

Section 3.6.4, "Page-Directory and Page-Table Entries," in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, will be clarified as follows:

If the processor generates a page-fault exception, the operating system must carry out the following operations in this order:

1. Copy the page from disk storage into physical memory if needed.
2. Load the page address into the page-table or page-directory entry and set its present flag. Other bits, such as the dirty and accessed bits, may also be set at this time.
3. Invalidate the current page table entry in the TLB (see Section 3.7, "Translation Lookaside Buffers (TLBs)" for a discussion of TLBs and how to invalidate them).
4. Return from the page fault handler to restart the interrupted program or task.

4. Performance-Monitoring Counter Issues

The following table documents the characterized differences between the behavior of the Intel Celeron processor's performance-monitoring counters and that documented in Appendix A of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*.

The following table replaces Table A-1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. The only change to this new table are enhanced descriptions of the events counted.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_ MEM_ REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only external memory loads and stores, but also internal retries. Note: 80 bit floating point accesses are double counted, since they are decomposed into a 16 bit exponent load and a 64 bit mantissa load. Memory accesses are only counted when they are actually performed. E.g., a load that gets squashed because a previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once. Does not include I/O accesses, or other nonmemory accesses.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	45H	DCU_LINES_IN	00H	Total lines allocated in the DCU.	
	46H	DCU_M_LINES_IN	00H	Number of M state lines allocated in the DCU.	
	47H	DCU_M_LINES_OUT	00H	Number of M state lines evicted from the DCU. This includes evictions via snoop HITM, intervention or replacement.	
	48H	DCU_MISS_OUTSTANDING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are considered. Uncacheable requests are excluded. Read-for-ownerships are counted as well as line fills, invalidates, and stores.	An access that also misses the L2 is short-changed by 2 cycles. (i.e., if count is N cycles, should be N+2 cycles.) Subsequent loads to the same cache line will not result in any additional counts. Count value not precise, but still useful.
Instruction Fetch Unit (IFU)	80H	IFU_IFETCH	00H	Number of instruction fetches, both cacheable and noncacheable. Including UC fetches.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	81H	IFU_ IFETCH_ MISS	00H	Number of instruction fetch misses. All instruction fetches that do not hit the IFU, i.e., that produce memory requests. Includes UC accesses.	
	85H	ITLB_ MISS	00H	Number of ITLB misses.	
	86H	IFU_ MEM_ STALL	00H	Number of cycles instruction fetch is stalled, for any reason. Includes IFU cache misses, ITLB misses, ITLB faults and other minor stalls.	
	87H	ILD_ STALL	00H	Number of cycles that the instruction length decoder is stalled.	
L2 Cache[1]	28H	L2_ IFETCH	MESI 0FH	Number of L2 instruction fetches. This event indicates that a normal instruction fetch was received by the L2. The count includes only L2 cacheable instruction fetches; it does not include UC instruction fetches. It does not include ITLB miss accesses.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	29H	L2_LD	MESI 0FH	Number of L2 data loads. This event indicates that a normal, unlocked, load memory access was received by the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It does include L2 cacheable TLB miss memory accesses.	
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2. Specifically, it indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable store memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses like	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				UC/WT stores. It includes TLB miss memory accesses.	
	24H	L2_LINES_IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_OUT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_LINES_INM	00H	Number of modified lines allocated in the L2.	
	27H	L2_M_LINES_OUTM	00H	Number of modified lines removed from the L2 for any reason.	
	2EH	L2_RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_BUSY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_BUSY_RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
External Bus Logic (EBL)[2]	62H	BUS_ DRDY_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which DRDY# is asserted. Essentially, utilization of the external system data bus.	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#. Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.
	63H	BUS_ LOCK_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus.	Always counts in processor clocks.
	60H	BUS_REQ _OUT-STAND-ING	00H (Self)	Number of bus requests outstanding. This counter is incremented by the number of cacheable read bus requests outstanding in any given cycle.	Counts only DCU full-line cacheable reads, not RFOs, writes, instruction fetches, or anything else. Counts “waiting for bus to complete” (last data chunk received).
	65H	BUS_ TRAN_ BRD	00H (Self) 20H (Any)	Number of burst read transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	66H	BUS_ TRAN_ RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_ TRANS_W B	00H (Self) 20H (Any)	Number of completed write back transactions.	
	68H	BUS_ TRAN_ IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	
	69H	BUS_ TRAN_ INVAL	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_ TRAN_ PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_ TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_ TRANS_ IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_ TRAN_ DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	
	6EH	BUS_ TRAN_ BURST	00H (Self) 20H (Any)	Number of completed burst transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	70H	BUS_ TRAN_ ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles, etc.	
	6FH	BUS_ TRAN_ MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	
	64H	BUS_ DATA_ RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR _DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	
	7AH	BUS_HIT_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HIT# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1,

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
					and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7BH	BUS_ HITM_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
					set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7EH	BUS_ SNOOP_ STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	
Floating Point Unit	C1H	FLOPS	00H	Number of computational floating-point operations retired. Excludes floating point computational operations that cause traps or assists. Includes floating point computational operations executed by the assist handler. Includes internal sub-operations of complex floating	Counter 0 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				point instructions like transcendentals. Excludes floating point loads and stores.	
	10H	FP_COMP_OPS_EXE	00H	Number of computational floating-point operations executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs and IDIVs. Note not the number of cycles but, the number of operations. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	Counter 0 only
	11H	FP_ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	Number of multiplies. Note: includes integer and well FP multiplies and is speculative.	Counter 1 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	13H	DIV	00H	Number of divides. Note: includes integer and FP multiplies and is speculative.	Counter 1 only
	14H	CYCLES_ DIV_ BUSY	00H	Number of cycles that the divider is busy, and cannot accept new divides. Note: includes integer and FP divides, FPREM, FPSQRT, e.g., and is speculative.	Counter 0 only
Memory Ordering	03H	LD_ BLOCKS	00H	Number of store buffer blocks. Includes counts caused by preceding stores whose addresses are unknown, preceding stores whose addresses are known to conflict, but whose data is unknown and preceding stores that conflicts with the load, but which incompletely overlap the load.	
	04H	SB_DRAINS	00H	Number of store buffer drain cycles. Incremented during every cycle the store buffer is draining. Draining is caused by serializing operations like CPUID,	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				synchronizing operations like XCHG, Interrupt acknowledgment as well as other conditions such as cache flushing.	
	05H	MIS-ALIGN_ MEM_ REF	00H	Number of misaligned data memory references. Incremented by 1 every cycle during which either the load or store pipeline dispatches a misaligned uop. Counting is performed if its the first half or second half, or if it is blocked, squashed or misses. Note in this context misaligned means crossing a 64 bit boundary.	It should be noted that MISALIGN_MEM_REF is only an approximation, to the true number of misaligned memory references. The value returned is roughly proportional to the number of misaligned memory accesses, i.e., the size of the problem.
Instr-uction Decod-ing and Retire ment	C0H	INST_ RETIRED	OOH	Number of instructions retired.	A hardware interrupt received during/after the last iteration of the REP STOS flow causes the counter to undercount by 1 instruction.
	C2H	UOPS_ RETIRED	00H	Number of UOPs retired.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	D0H	INST_ DECODER	00H	Number of instructions decoded.	
Interrupts	C8H	HW_INT_ RX	00H	Number of hardware interrupts received.	
	C6H	CYCLES_ INT_ MASKED	00H	Number of processor cycles for which interrupts are disabled.	
	C7H	CYCLES_ INT_ PENDING _AND_ MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Branches	C4H	BR_INST_ RETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_ PRED_ RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_ TAKEN_ RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_ PRED_ TAKEN_ RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_ DECODED	00H	Number of branch instructions decoded.	
	E2H	BTB_ MISSES	00H	Number of branches that for which the BTB did not produce a prediction	
	E4H	BR_ BOGUS	00H	Number of bogus branches.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	E6H	BACLEARS	00H	Number of time BACLEAR is asserted. This is the number of times that a static branch prediction was made, where the branch decoder decided to make a branch prediction because the BTB did not.	
Stalls	A2H	RESOURCE_STALLS	00H	Incremented by one during every cycle that there is a resource related stall. Includes register renaming buffer entries, memory buffer entries. Does not include stalls due to bus queue full, too many cache misses, e.g., In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, e.g., if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	D2H	PARTIAL_ RAT_ STALLS	00H	Number of cycles or events for partial stalls. Note Includes flag partial stalls.	
Segment Register Loads	06H	SEGMENT_ REG_ LOADS	00H	Number of segment register loads	
Clocks	79H	CPU_CLK _UNHALTED	00H	Number of cycles during which the processor is not halted.	

NOTES:

1. Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved. The Pentium® II processor identifies cache states using the "MESI" protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.
2. All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).

5. POP[ESP] with 16-bit Stack Size

In the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, the section regarding "POP–Pop a Value from the Stack," the following note:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register."

is incomplete, and should read as follows:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. For the case of a 16-bit stack where ESP wraps to 0h as a result of the POP instruction, the resulting location of the memory write is processor family specific."

In Section 17.23.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, add a new section:

A POP-to-memory instruction, which uses the stack pointer (ESP) as a base register.

For a POP-to-memory instruction that meets the following conditions:

1. The stack segment size is 16-bit.
2. Any 32-bit addressing form with the SIB byte specifying ESP as the base register.
3. The initial stack pointer is FFFCh(32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0h as a result of the POP operation.

The result of the memory write is processor family specific. For example, in Pentium II, Pentium Pro, and Intel Celeron processors, the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and i486™ processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 Kbyte), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbyte).